Tidy data - dplyr & tidyr

Lecture 08

Dr. Colin Rundel

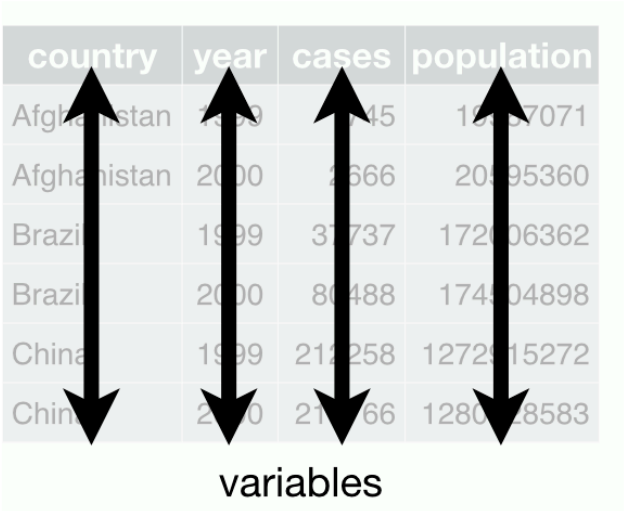

tidyverse
www.rstudio.com

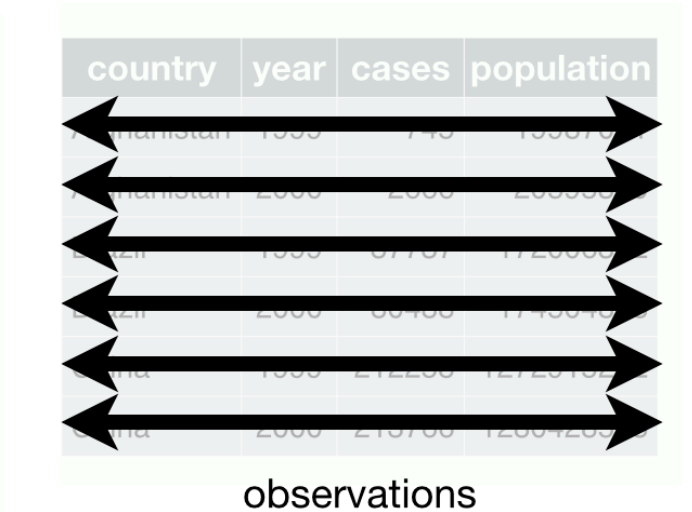
tidyverse package

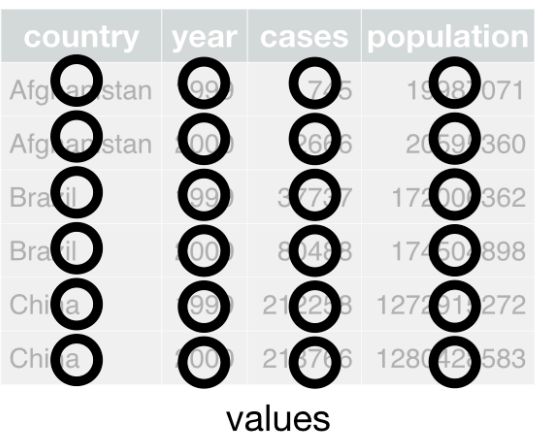
```
1 library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────── tidyverse 2.0.0 ──
✔ dplyr     1.2.0     ✔ readr     2.1.6
✔ forcats   1.0.1     ✔ stringr   1.6.0
✔ ggplot2   4.0.2     ✔ tibble    3.3.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.2
✔ purrr     1.2.1
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
✖ tidyr::extract()   masks magrittr::extract()
✖ dplyr::filter()    masks stats::filter()
✖ dplyr::lag()       masks stats::lag()
✖ purrr::set_names() masks magrittr::set_names()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force a
```

Tidy data


country
year
cases
population
variables


country
year
cases
population
observations


country
year
cases
population
values

Tidy vs Untidy

> Happy families are all alike; every unhappy family is unhappy in its own way
>
> — Leo Tolstoy, Anna Karenina

```
# A tibble: 317 × 7
   artist         track           date.entered   wk1   wk2   wk3   wk4
   <chr>          <chr>           <date>       <dbl> <dbl> <dbl> <dbl>
 1 2 Pac          Baby Don't Cry… 2000-02-26      87    82    72    77
 2 2Ge+her        The Hardest Pa… 2000-09-02      91    87    92    NA
 3 3 Doors Down   Kryptonite      2000-04-08      81    70    68    67
 4 3 Doors Down   Loser           2000-10-21      76    76    72    69
 5 504 Boyz       Wobble Wobble   2000-04-15      57    34    25    17
 6 98^0           Give Me Just O… 2000-08-19      51    39    34    26
 7 A*Teens        Dancing Queen   2000-07-08      97    97    96    95
 8 Aaliyah        I Don't Wanna   2000-01-29      84    62    51    41
 9 Aaliyah        Try Again       2000-03-18      59    53    38    28
10 Adams, Yolanda Open My Heart   2000-08-26      76    76    74    69
# ℹ 307 more rows
```

Is this data tidy?


TIBBLE
www.rstudio.com

Modern data frames

The tidyverse includes the tibble package that extends data frames to be a bit more modern. The core features of tibbles are a nicer printing method as well as being "lazy".

```
1 library(tibble)
```

```r
iris
```

```
    Sepal.Length Sepal.Width Petal.Length
1            5.1         3.5          1.4
2            4.9         3.0          1.4
3            4.7         3.2          1.3
4            4.6         3.1          1.5
5            5.0         3.6          1.4
6            5.4         3.9          1.7
7            4.6         3.4          1.4
8            5.0         3.4          1.5
9            4.4         2.9          1.4
10           4.9         3.1          1.5
11           5.4         3.7          1.5
12           4.8         3.4          1.6
13           4.8         3.0          1.4
14           4.3         3.0          1.1
15           5.8         4.0          1.2
16           5.7         4.4          1.5
17           5.4         3.9          1.3
18           5.1         3.5          1.4
19           5.7         3.8          1.7
20           5.1         3.8          1.5
21           5.4         3.4          1.7
22           5.1         3.7          1.5
```

```r
(tbl_iris = as_tibble(iris))
```

```
# A tibble: 150 × 5
   Sepal.Length Sepal.Width Petal.Length
          <dbl>       <dbl>        <dbl>
 1          5.1         3.5          1.4
 2          4.9         3            1.4
 3          4.7         3.2          1.3
 4          4.6         3.1          1.5
 5          5           3.6          1.4
 6          5.4         3.9          1.7
 7          4.6         3.4          1.4
 8          5           3.4          1.5
 9          4.4         2.9          1.4
10          4.9         3.1          1.5
# ℹ 140 more rows
# ℹ 2 more variables: Petal.Width <dbl>,
#   Species <fct>
```

Tibbles are lazy (preserving type)

By default, subsetting tibbles always results in another tibble (`$` or `[[` can still be used to subset for a specific column).

```
1 tbl_iris[1,]
```

```
# A tibble: 1 × 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.1         3.5          1.4         0.2 setosa
```

```r
tbl_iris[,1]
```

```
# A tibble: 150 × 1
   Sepal.Length
          <dbl>
 1          5.1
 2          4.9
 3          4.7
 4          4.6
 5          5
 6          5.4
 7          4.6
 8          5
 9          4.4
10          4.9
# ℹ 140 more rows
```

```r
head(tbl_iris[[1]])
```

```
[1] 5.1 4.9 4.7 4.6 5.0 5.4
```

```r
head(tbl_iris$Species)
```

```
[1] setosa setosa setosa setosa setosa set
Levels: setosa versicolor virginica
```

Tibbles are lazy (partial matching)

Tibbles do not use partial matching when the `$` operator is used.

```
1 head( iris$Species )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( tbl_iris$Species )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( iris$Sp )
```

```
[1] setosa setosa setosa setosa
Levels: setosa versicolor virgir
```

```
1 head( tbl_iris$Sp )
```

```
Warning: Unknown or uninitialise
NULL
```

Tibbles are lazy (length coercion)

Only vectors with length 1 will undergo length coercion / recycling - anything else throws an error.

```
1 data.frame(x = 1:4, y = 1)
```

```
  x y
1 1 1
2 2 1
3 3 1
4 4 1
```

```
1 tibble(x = 1:4, y = 1)
```

```
# A tibble: 4 × 2
      x     y
  <int> <dbl>
1     1     1
2     2     1
3     3     1
4     4     1
```

```
1 data.frame(x = 1:4, y = 1:2)
```

```
  x y
1 1 1
2 2 2
3 3 1
4 4 2
```

```
1 tibble(x = 1:4, y = 1:2)
```

```
Error in `tibble()`:
! Tibble columns must have compatib
• Size 4: Existing data.
• Size 2: Column `y`.
ℹ Only values of size one are recyc
```

Tibbles and S3

```r
t = tibble(
  x = 1:3,
  y = c("A","B","C")
)

class(t)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```r
d = data.frame(
  x = 1:3,
  y = c("A","B","C")
)

class(d)
```

```
[1] "data.frame"
```

```r
methods(class="tbl_df")
```

```
 [1] [             [[            [[<-          [<-           %within%
 [6] $             $<-           as.data.frame cbind2        coerce
[11] fortify       group_data    initialize    kronecker     names<-
[16] nest_legacy   nest          Ops           rbind2        row.names<-
[21] show          slotsFromS3   str           tbl_sum
see '?methods' for accessing help and source code
```

```r
methods(class="tbl")
```

```
 [1] [[<-        [<-         %within%    $<-         cbind2
 [6] coerce      format      fortify     glimpse     initialize
[11] kronecker   Ops         print       rbind2      show
[16] slotsFromS3 tbl_sum
see '?methods' for accessing help and source code
```

Tibble support?

Tibbles are just specialized data frames, and will fall back to base data frame methods when needed.

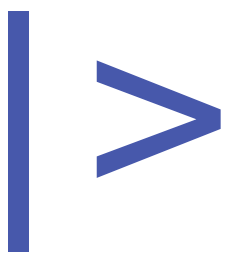
```
d = tibble(
  x = rnorm(100),
  y = 3 + x + rnorm(100, sd = 0.1)
)
```

```
lm(y~x, data = d)
```

```
Call:
lm(formula = y ~ x, data = d)

Coefficients:
(Intercept)            x
     3.0041       0.9967
```

Why did this work?



R’s base pipe

What is a pipe

> In software engineering, a pipeline consists of a chain of processing elements (processes, threads, coroutines, functions, etc.), arranged so that the output of each element is the input of the next;
>
> Wikipedia - Pipeline (software)

R's base pipe (`|>`) is an infix operator that allows us to link two functions together in a way that is readable from left to right.

The following examples are equivalent:

```
1 f(g(x=1, y=2), n=2)
```

```
1 g(x=1, y=2) |> f(n=2)
```

```
1 g(x=1, y=2) %>% f(n=2)  # library(magrittr)
```

Readability

All of the following are fine; it comes down to personal preference. Try to be consistent.

Nested:

```
h( g( f(x), y=1), z=1 )
```

Piped:

```
f(x) |>
  g(y=1) |>
  h(z=1)
```

Intermediate:

```
res = f(x)
res = g(res, y=1)
res = h(res, z=1)
```

What about other arguments?

Sometimes we want to send our results to a function argument other than first one. In these cases we can refer to the previous result using _ (as long as the argument is named).

```
1 data.frame(a = 1:3, b = 3:1) |> lm(a~b, data=_)
```

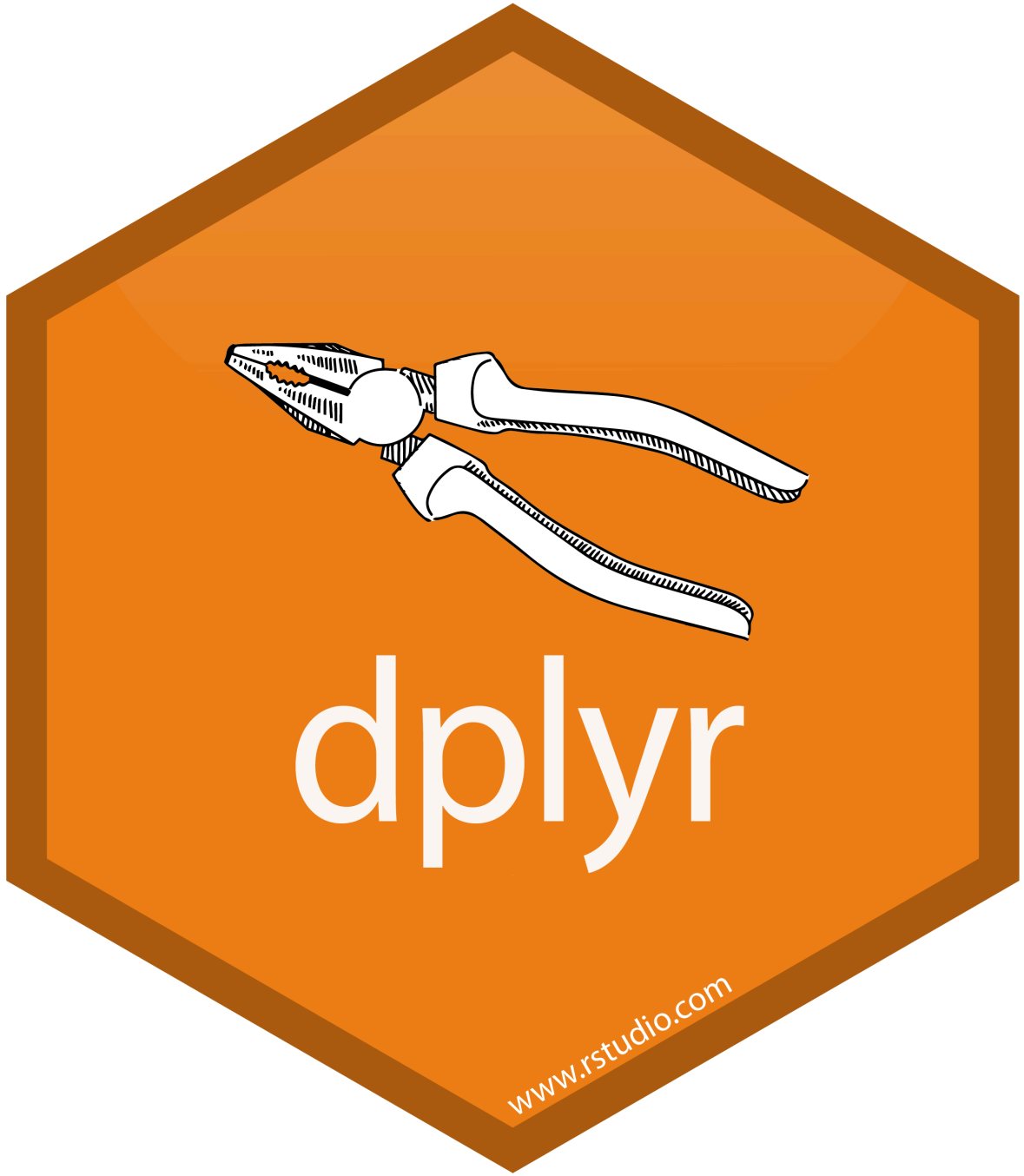
```
Call:
lm(formula = a ~ b, data = data.frame(a = 1:3, b = 3:1))

Coefficients:
(Intercept)            b
          4           -1
```

For non-named arguments we can use an anonymous function.

```
1 data.frame(a = 1:3, b = 3:1) |> (function(x) x[[1]])()
```

```
[1] 1 2 3
```


dplyr
www.rstudio.com

A Grammar of Data Manipulation

dplyr is based on the concepts of functions as verbs that manipulate data frames.

Core single data frame functions / verbs:

- `filter()` / `slice()` - pick rows based on criteria
- `mutate()` / `transmute()` - create or modify columns
- `summarise()` / `count()` - reduce variables to values
- `group_by()` / `ungroup()` - modify other verbs to act on subsets
- `select()` / `rename()` - select columns by name
- `pull()` - grab a column as a vector
- `arrange()` - reorder rows
- `distinct()` - filter for unique rows
- `relocate()` - change column order
- … (many more)

dplyr rules

1. First argument is *always* a data frame
2. Subsequent arguments say what to do with the data frame
3. *Always* return a data frame
4. Don’t modify in place (how does this affect memory usage?)
5. Magic via non-standard evaluation + lazy evaluation and S3

Example Data

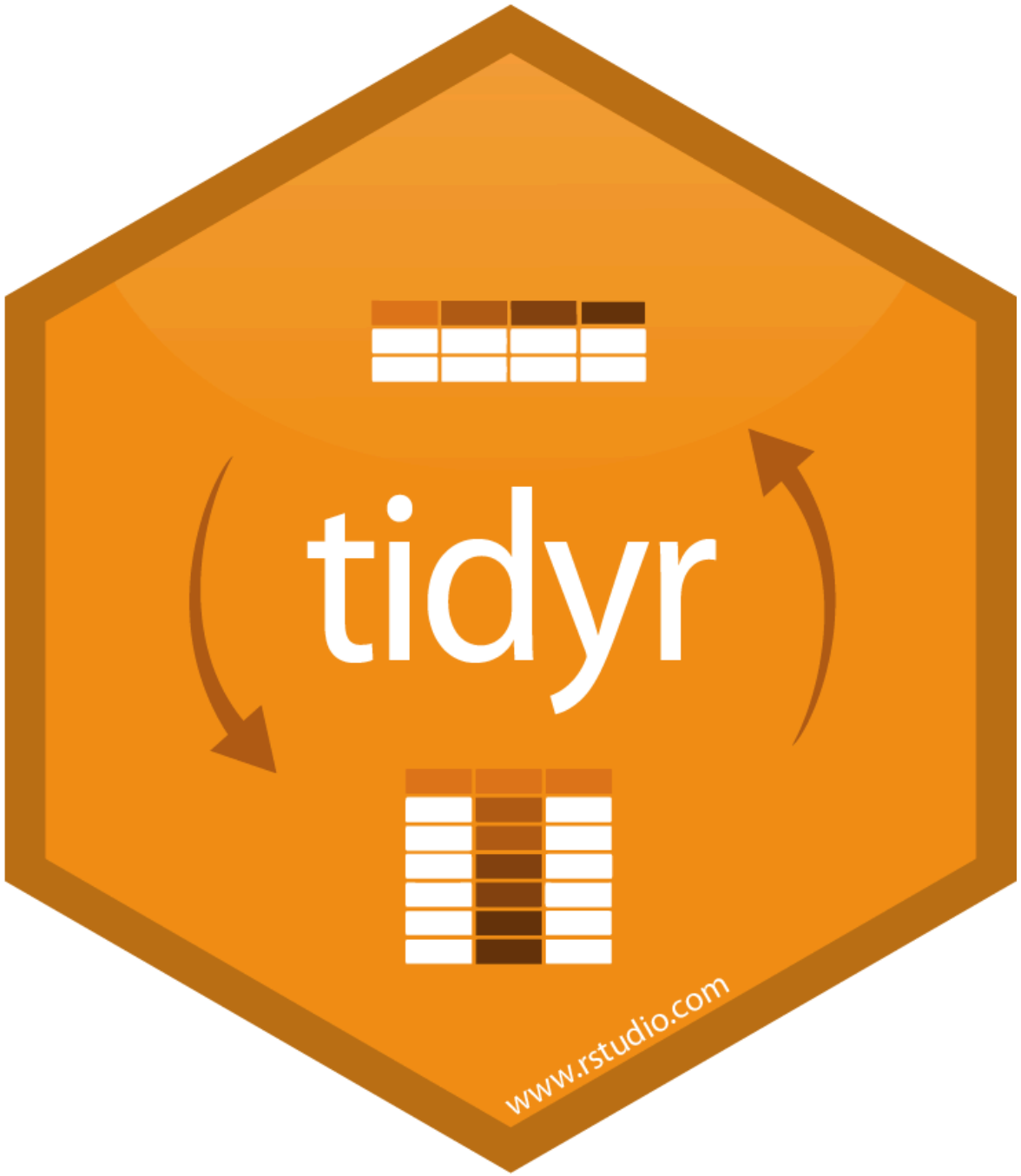
We will demonstrate dplyr's functionality using the nycflights13 data.

```
1 library(nycflights13)
2 flights
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ℹ 336,766 more rows
# ℹ 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>
```

Example 1

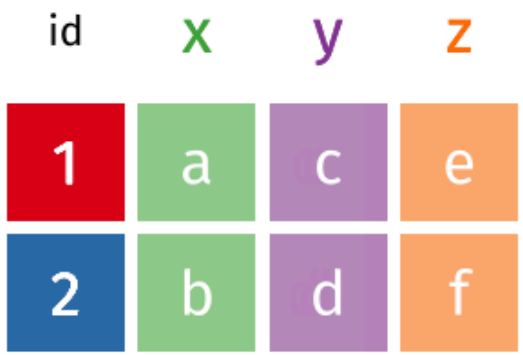
1. How many flights to Los Angeles (LAX) did each of the legacy carriers (AA, UA, DL or US) have in May from JFK, and what was their average duration?
2. Which plane (check the tail numbers) flew out of each New York airport the most?
3. Which 5 days should you consider flying on if you want to have the lowest possible average departure delay?
4. Which flight has the largest arrival delay as a percentage of its scheduled air time?


tidyr
www.rstudio.com

Reshaping data

Wide vs Long

wide



Wide -> Long

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

`pivot_longer` (previously `gather` or `reshape2::melt`)

Syntax

```
(d = tibble::tribble(
  ~country, ~"1999",  ~"2000",
        "A", "0.7K",     "2K",
        "B",  "37K",    "80K",
        "C", "212K",   "213K"
))
```

```
# A tibble: 3 × 3
  country `1999` `2000`
  <chr>   <chr>  <chr>
1 A       0.7K   2K
2 B       37K    80K
3 C       212K   213K
```

```
pivot_longer(
  d,
  cols = "1999":"2000",
  names_to = "year",
  values_to = "cases"
)
```

```
# A tibble: 6 × 3
  country year  cases
  <chr>   <chr> <chr>
1 A       1999  0.7K
2 A       2000  2K
3 B       1999  37K
4 B       2000  80K
5 C       1999  212K
6 C       2000  213K
```

Long -> Wide

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

`pivot_wider` (previously `spread`)

Syntax

```
( d = tibble::tribble(
    ~country, ~year,   ~type, ~count,
         "A",  1999, "cases", "0.7K",
         "A",  1999,   "pop",  "19M",
         "A",  2000, "cases",   "2K",
         "A",  2000,   "pop",  "20M",
         "B",  1999, "cases",  "37K",
         "B",  1999,   "pop", "172M",
         "B",  2000, "cases",  "80K",
         "B",  2000,   "pop", "174M",
         "C",  1999, "cases", "212K",
         "C",  1999,   "pop",   "1T",
         "C",  2000, "cases", "213K",
         "C",  2000,   "pop",   "1T"
  )
)
```

```
# A tibble: 12 × 4
   country  year type  count
   <chr>   <dbl> <chr> <chr>
 1 A        1999 cases 0.7K
 2 A        1999 pop   19M
 3 A        2000 cases 2K
 4 A        2000 pop   20M
 5 B        1999 cases 37K
 6 B        1999 pop   172M
 7 B        2000 cases 80K
 8 B        2000 pop   174M
 9 C        1999 cases 212K
10 C        1999 pop   1T
11 C        2000 cases 213K
12 C        2000 pop   1T
```

```
pivot_wider(
  d,
  id_cols = country:year,
  names_from = type,
  values_from = count
)
```

```
# A tibble: 6 × 4
  country  year cases pop
  <chr>   <dbl> <chr> <chr>
1 A        1999 0.7K  19M
2 A        2000 2K    20M
3 B        1999 37K   172M
4 B        2000 80K   174M
5 C        1999 212K  1T
6 C        2000 213K  1T
```

Exercise 1

The `palmerpenguins` package contains measurement data on various penguin species on islands near Palmer Station in Antarctica. The code below shows the # of each species measured on each of the three islands (missing island, penguin pairs implies that species does not occur on that island).

```
palmerpenguins::penguins |>
  count(island, species)
```

```
# A tibble: 5 × 3
  island    species       n
  <fct>     <fct>     <int>
1 Biscoe    Adelie       44
2 Biscoe    Gentoo      124
3 Dream     Adelie       56
4 Dream     Chinstrap    68
5 Torgersen Adelie       52
```

Starting from these data construct a contingency table of counts for island (rows) by species (columns) using the pivot functions we've just discussed.

Separate - wider

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

```
1 separate_wider_delim(d, rate, delim = "/", names = c("cases", "pop"))
```

```
# A tibble: 6 × 4
  country  year cases pop
  <chr>   <dbl> <chr> <chr>
1 A        1999 0.7K  19M
2 A        2000 2K    20M
3 B        1999 37K   172M
4 B        2000 80K   174M
5 C        1999 212K  1T
6 C        2000 213K  1T
```

Separate - longer

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

```
1 separate_longer_delim(d, rate, delim =
```

```
# A tibble: 12 × 3
   country  year rate
   <chr>   <dbl> <chr>
 1 A        1999 0.7K
 2 A        1999 19M
 3 A        2000 2K
 4 A        2000 20M
 5 B        1999 37K
 6 B        1999 172M
 7 B        2000 80K
 8 B        2000 174M
 9 C        1999 212K
10 C        1999 1T
11 C        2000 213K
12 C        2000 1T
```

Other separates

In previous versions of tidyr there was a single catch-all `separate()` function. This still exists and is available in the package but it is **superseded**.

Other helpful separate functions:

- `separate_longer_position()`
- `separate_wider_position()`
- `separate_wider_regex()`

Unite

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

```
1 unite(d, century, year, col = "year", sep = "")
```

```
# A tibble: 6 × 2
  country year
  <chr>   <chr>
1 Afghan  1999
2 Afghan  2000
3 Brazil  1999
4 Brazil  2000
5 China   1999
6 China   2000
```

Example 2 - tidy grades

Is the following data tidy?

```
grades = tibble::tribble(
  ~name,   ~hw_1, ~hw_2, ~hw_3, ~hw_4, ~proj_1, ~proj_2,
  "Alice",    19,    19,    18,    20,      89,      95,
  "Bob",      18,    20,    18,    16,      77,      88,
  "Carol",    18,    20,    18,    17,      96,      99,
  "Dave",     19,    19,    18,    19,      86,      82
)
```

How would we calculate a final score based on the following formula,

$$\text{score} = 0.5 \frac{\sum_i \text{hw}_i}{80} + 0.5 \frac{\sum_j \text{proj}_j}{200}$$

Semi-tidy approach

```
grades |>
  mutate(
    hw_avg = (hw_1+hw_2+hw_3+hw_4)/4,
    proj_avg = (proj_1+proj_2)/2
  ) |>
  mutate(
    overall = 0.5*(proj_avg/100) + 0.5*(hw_avg/20)
  )
```

```
# A tibble: 4 × 10
  name   hw_1  hw_2  hw_3  hw_4 proj_1 proj_2 hw_avg proj_avg overall
  <chr> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>    <dbl>   <dbl>
1 Alice    19    19    18    20     89     95   19        92     0.935
2 Bob      18    20    18    16     77     88   18        82.5   0.862
3 Carol    18    20    18    17     96     99   18.2      97.5   0.944
4 Dave     19    19    18    19     86     82   18.8      84     0.889
```

pivot_longer (Wide -> Long)

```
tidyr::pivot_longer(
  grades,
  cols = hw_1:proj_2,
  names_to = "assignment",
  values_to = "score"
)
```

```
# A tibble: 24 × 3
   name  assignment score
   <chr> <chr>      <dbl>
 1 Alice hw_1          19
 2 Alice hw_2          19
 3 Alice hw_3          18
 4 Alice hw_4          20
 5 Alice proj_1        89
 6 Alice proj_2        95
 7 Bob   hw_1          18
 8 Bob   hw_2          20
 9 Bob   hw_3          18
10 Bob   hw_4          16
# ℹ 14 more rows
```

Split type and id

```
tidyr::pivot_longer(
  grades,
  cols = hw_1:proj_2,
  names_to = c("type", "id"),
  names_sep = "_",
  values_to = "score"
)
```

```
# A tibble: 24 × 4
   name  type  id    score
   <chr> <chr> <chr> <dbl>
 1 Alice hw    1        19
 2 Alice hw    2        19
 3 Alice hw    3        18
 4 Alice hw    4        20
 5 Alice proj  1        89
 6 Alice proj  2        95
 7 Bob   hw    1        18
 8 Bob   hw    2        20
 9 Bob   hw    3        18
10 Bob   hw    4        16
# ℹ 14 more rows
```

Tidy approach?

```
grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  )
```

```
# A tibble: 8 × 3
  name  type  total
  <chr> <chr> <dbl>
1 Alice hw       76
2 Alice proj    184
3 Bob   hw       72
4 Bob   proj    165
5 Carol hw       73
6 Carol proj    195
7 Dave  hw       75
8 Dave  proj    168
```

pivot_wider - (Long -> Wide)

```
grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  ) |>
  tidyr::pivot_wider(
    names_from = type,
    values_from = total
  )
```

```
# A tibble: 4 × 3
  name     hw  proj
  <chr> <dbl> <dbl>
1 Alice    76   184
2 Bob      72   165
3 Carol    73   195
4 Dave     75   168
```

Wrapping up

```
final_grades = grades |>
  tidyr::pivot_longer(
    cols = hw_1:proj_2,
    names_to = c("type", "id"),
    names_sep = "_",
    values_to = "score"
  ) |>
  summarize(
    total = sum(score),
    .by = c(name, type)
  ) |>
  tidyr::pivot_wider(
    names_from = type,
    values_from = total
  ) |>
  mutate(
    score = 0.5*(hw/80) +
            0.5*(proj/200)
  )

```

```
# A tibble: 4 × 4
  name     hw  proj score
  <chr> <dbl> <dbl> <dbl>
1 Alice    76   184 0.935
2 Bob      72   165 0.862
3 Carol    73   195 0.944
4 Dave     75   168 0.889
```